Follow-up Assignment:
Encoding and Persistent Storage of Graphical Data in Java

Part 2

November 18, 2025

1. Attach the file `MyFrame.java`, supplied with the assignment, to the verified and tested project containing the file `PositionAndColor.java`. Check whether the program starts and works in its basic form.
2. Analyze the provided code and see how the class `PositionAndColor.java` is used in it.
3. Next, choose from the menu: *View -> Tool Windows -> TODO* and, based on the names, complete the missing code fragments:
   - Define a private instance method named `saveInt`, taking an output stream `OutputStream os` and an integer value `int value`, which may throw an `IOException` and is intended to save this value as four bytes to the stream:
     - Write the first byte of the value to the stream by calling the `write` method on the `os` object with the argument `value`, so that the least significant byte of the integer is sent to the stream.
     - Write the second byte of the value to the stream by calling the `write` method on the `os` object with the argument `value`, so that after shifting the number by 8 positions to the right, the next group of eight bits is sent to the stream.
     - Write the third byte of the value to the stream by calling the `write` method on the `os` object with the argument `value`, so that after another shift by 16 bits, the next byte of the number is stored in the stream.
     - Write the fourth byte of the value to the stream by calling the `write` method on the `os` object with the argument `value`, in order to complete the writing of four bytes representing the integer in order from the least significant to the most significant byte.
   - Define a private instance method named `loadInt`, which takes an input stream `InputStream is`, may throw an `IOException`, and returns an `int` value reconstructed from four consecutive bytes read from the stream:
     - Read the first byte from the stream by calling `read` on the `is` object, assign the returned value to a local variable `value` of type `int`, and treat it as the least significant byte of the integer being constructed.
     - Read the second byte from the stream, again by calling the `read` method on the `is` object, shift the obtained value left by 8 bits and append it to the variable

    `value` using the bitwise OR operation, so that this second portion of data occupies the next eight bits of the number.

  – Read the third byte from the stream by calling `read` on the `is` object, shift the read value left by 16 bits and combine it with the current value of `value` using the bitwise OR operation, thus placing the third byte in the appropriate part of the number.
  – Read the fourth byte from the stream by calling `read` on the `is` object, shift the result left by 24 bits and append it to the variable `value` with the bitwise OR operation, and then return from the method the value `value` as a complete integer reconstructed from four consecutive bytes of data.

- Implement the event handler method `actionPerformed` for the `saveButton`, which takes a parameter of type `ActionEvent` and will be responsible for saving the current state of the circles to a binary file.
  – In the body of the method, open an output stream to a binary file using `FileOutputStream`, placing its creation in the header of a `try` block with the try with resources construct, so that the stream is automatically closed after the write operations finish.
  – Inside the `try` block, write to the stream information about the number of circles by calling the method `MyFrame.this.saveInt` with the open stream and the field `MyFrame.this.circlesCount` as arguments, so that the file stores the number of elements that will be written next.
  – After writing the number of circles, iterate over all elements of the collection or array `MyFrame.this.circles` with a for each loop, and for each value call the method `MyFrame.this.saveInt` with the open stream and the current element, so that all encoded representations of circles are written sequentially to the binary file.
  – Handle potential input output exceptions by adding `catch` blocks for `FileNotFoundException` and `IOException`, so that errors related to file access or data writing are clearly signaled during program execution.
- Implement the event handler method `actionPerformed` for the `loadButton`, which takes a parameter of type `ActionEvent` and will be responsible for restoring the state of the circles from a binary file into the data structure stored in the `MyFrame` object.
  – In the body of the method, open an input stream to the binary file using `FileInputStream`, placing its creation in the header of a `try` block with the try with resources construct, so that the stream is automatically closed after the read operations finish, regardless of whether the subsequent steps succeed.
  – Inside the `try` block, read from the file the number of saved circles:
    ∗ Call the method `MyFrame.this.loadInt`, passing the open input stream `fis` as an argument.
    ∗ Assign the returned value to the field `MyFrame.this.circlesCount` to update the number of elements that should then be read into the circles array.

  - Read from the file the subsequent encoded representations of circles and store them in the array:
    * Create a loop iterating over indices from zero to the value `MyFrame.this.circlesCount` minus one.
    * In each iteration, call the method `MyFrame.this.loadInt` with the argument `fis`, and assign the returned value to the element of the array `MyFrame.this.circles` at the current index, thereby reconstructing the contents of the array from the data stored in the binary file.
  - Handle potential input output errors by adding `catch` blocks for the exceptions `FileNotFoundException` and `IOException`, in order to signal problems with file access or data reading during program execution.
- Check the behavior of the program, in particular the saving and reading of information, and then present the result to the instructor.

4. Transform the methods related to `loadButton` and `saveButton` so that they use the approach implemented in the `nio` package:

   - In the `actionPerformed` method for the `saveButton`, open a file channel using the `FileChannel.open` method, passing the path to the binary file with the circles and appropriate write options, so that the file is created if it does not exist, opened in write mode, and cleared before the content is written again.
     - Compute the size of the buffer needed to hold all the data by multiplying the constant `Integer.BYTES` by the sum of one (for saving the number of circles) and the length of the array `MyFrame.this.circles`, and then create a `ByteBuffer` buffer of the calculated size using the `allocate` method.
     - Write to the buffer the information about the number of circles by calling the method `putInt` on the buffer object with the argument `MyFrame.this.circlesCount`, so that the first value in the buffer represents the number of subsequent entries.
     - Iterate over all elements of the array `MyFrame.this.circles` using a for each loop and, for each element, call the method `putInt` on the buffer to place all encoded representations of the circles in it sequentially.
     - Prepare the buffer for the write operation to the channel by calling the `flip` method, and then send the contents of the buffer to the file by calling the `write` method on the `channel` object with the buffer as an argument.
     - Handle possible input output errors by surrounding the entire operation with a `try` block and a `catch` clause for the `IOException` exception, so that problems with writing to the file are signaled.
   - In the `actionPerformed` method for the `loadButton`, open a file channel using the `FileChannel.open` method, passing the path to the file with the circles and the option `StandardOpenOption.READ`, placing the creation of the channel in the header of a

`try` block with the try with resources construct, so that the channel is automatically closed after the read operations finish.

  - Determine the file size by calling the `size` method on the channel object and storing the result in a local variable of type `long`, and then create a `ByteBuffer` buffer with a size equal to the file size by converting this value to `int` and passing it to the `allocate` method.
  - Read the entire contents of the file into the buffer by calling the `read` method on the channel object with the prepared buffer as an argument, and then prepare the buffer for reading from its beginning by calling the `flip` method on it.
  - Read from the buffer the first integer using the `getInt` method and assign it to the field `MyFrame.this.circlesCount` to obtain information about the number of saved circles, and then create a new `int` array with the length equal to this value and assign it to the field `MyFrame.this.circles`.
  - Iterate in a loop over indices from zero to the value `MyFrame.this.circlesCount` minus one, and in each iteration read the next integer from the buffer using `getInt`, assigning it to the element of the array `MyFrame.this.circles` at the current index, in order to reconstruct all encoded circles from the file.
  - Handle possible input output errors by surrounding the entire code fragment with a `try` block and a `catch` clause for the `IOException` exception, in order to signal a file read problem during program execution.

- Check the behavior of the program, in particular the saving and reading of information, and then present the result to the instructor.